The Transportation Problem:

Milk in a milk shed area is collected on three routes A, B and C. There are four chilling centers P, Q, R and S where milk is kept before transporting it to a milk plant. Each route is able to supply on an average one thousand liters of milk per day. The supply of milk on routes A, B and C are 150, 160 and 90 thousand liters respectively. Daily capacity in thousand liters of chilling centers is 140, 120, 90 and 50 respectively. The cost of transporting 1000 liters of milk from each route (source) to each chilling center (destination) differs according to the distance. These costs (in Rs.) are shown in the following table.

Routes	Chilling centers			
	P	Q	R	S
A	16	18	21	12
B	17	19	14	13
C	32	11	15	10

The problem is to determine how many thousand liters of milk is to be transported from each route on daily basis in order to minimize the total cost of transportation.

In [16]:
```
using JuMP, GLPK, DelimitedFiles
```

In [17]:
```
# Reading the data file and preparting arrays
data_file = "transportation-milk.csv"
data = readdlm(data_file, ',')
```

Out[17]:
```
5×6 Array{Any,2}:
    ""   ""   140     120     90     50
    ""   ""      "P"     "Q"    "R"    "S"
 150    "A"   16      18      21     12
 160    "B"   17      19      14     13
  90    "C"   32      11      15     10
```

In [18]:
```
supply_nodes = data[3:end, 2]
s = data[3:end, 1]
```

Out[18]:
```
3-element Array{Any,1}:
 150
 160
  90
```

In [19]:
```
demand_nodes = collect(data[2, 3:end])
d = collect(data[1, 3:end])
```

Out[19]:
```
4-element Array{Any,1}:
 140
 120
  90
  50
```

In [20]:
```
c = data[3:end, 3:end]
```

Out[20]:
```
3×4 Array{Any,2}:
 16  18  21  12
 17  19  14  13
 32  11  15  10
```

In [21]:
```
# Converting arrays to dictionaries
s_dict = Dict(supply_nodes .=> s)
d_dict = Dict(demand_nodes .=> d)
```

Out[21]:
```
Dict{SubString{String},Int64} with 4 entries:
  "Q" => 120
  "S" => 50
  "P" => 140
  "R" => 90
```

In [22]:
```
c_dict = Dict()
for i in 1:length(supply_nodes)
  for j in 1:length(demand_nodes)
    c_dict[supply_nodes[i], demand_nodes[j]] = c[i,j]
  end
end
```

In [23]:
```
# Preparing an Optimization Model
tp = Model(with_optimizer(GLPK.Optimizer))
```

Out[23]:

feasibility
Subject to

In [24]:
```
@variable(tp, x[supply_nodes, demand_nodes] >= 0)
```

Out[24]:
```
2-dimensional DenseAxisArray{VariableRef,2,...} with index sets:
    Dimension 1, Any["A", "B", "C"]
    Dimension 2, Any["P", "Q", "R", "S"]
And data, a 3×4 Array{VariableRef,2}:
 x[A,P]  x[A,Q]  x[A,R]  x[A,S]
 x[B,P]  x[B,Q]  x[B,R]  x[B,S]
 x[C,P]  x[C,Q]  x[C,R]  x[C,S]
```

In [25]:
```
@objective(tp, Min, sum(c_dict[i,j]*x[i,j]
                    for i in supply_nodes, j in demand_nodes))
```

Out[25]:

$$16x_{A,P} + 18x_{A,Q} + 21x_{A,R} + 12x_{A,S} + 17x_{B,P} + 19x_{B,Q} + 14x_{B,R} + 13x_{B,S} + 32x_{C,P} + 1$$

In [26]:
```
for i in supply_nodes
  @constraint(tp, sum(x[i,j] for j in demand_nodes) == s_dict[i] )
end
```

In [27]:
```
for j in demand_nodes
  @constraint(tp, sum(x[i,j] for i in supply_nodes) == d_dict[j] )
end
```

In [28]:
```
print(tp)
```

```
Min 16 x[A,P] + 18 x[A,Q] + 21 x[A,R] + 12 x[A,S] + 17 x[B,P] + 19 x[B,
Q] + 14 x[B,R] + 13 x[B,S] + 32 x[C,P] + 11 x[C,Q] + 15 x[C,R] + 10 x
[C,S]
Subject to
 x[A,P] + x[A,Q] + x[A,R] + x[A,S] = 150.0
 x[B,P] + x[B,Q] + x[B,R] + x[B,S] = 160.0
 x[C,P] + x[C,Q] + x[C,R] + x[C,S] = 90.0
 x[A,P] + x[B,P] + x[C,P] = 140.0
 x[A,Q] + x[B,Q] + x[C,Q] = 120.0
 x[A,R] + x[B,R] + x[C,R] = 90.0
 x[A,S] + x[B,S] + x[C,S] = 50.0
 x[A,P] ≥ 0.0
 x[A,Q] ≥ 0.0
 x[A,R] ≥ 0.0
 x[A,S] ≥ 0.0
 x[B,P] ≥ 0.0
 x[B,Q] ≥ 0.0
 x[B,R] ≥ 0.0
 x[B,S] ≥ 0.0
 x[C,P] ≥ 0.0
 x[C,Q] ≥ 0.0
 x[C,R] ≥ 0.0
 x[C,S] ≥ 0.0
```

In [29]:
```
optimize!(tp)
```

In [30]:
```
JuMP.value.(x)
```

Out[30]:
```
2-dimensional DenseAxisArray{Float64,2,...} with index sets:
    Dimension 1, Any["A", "B", "C"]
    Dimension 2, Any["P", "Q", "R", "S"]
And data, a 3×4 Array{Float64,2}:
 140.0   0.0   0.0  10.0
   0.0  30.0  90.0  40.0
   0.0  90.0   0.0   0.0
```

In [ ]: